Product Requirements Document: Meal Randomizer App

1. Elevator Pitch

The Meal Randomizer App is a smart, flexible tool that simplifies meal planning for any eating style. Users can create custom meal types—like "Breakfast," "Lunch," or "Snack"—and define unique categories and food lists for each. By default, pressing the "Generate Meal" button pulls one random item from every category for the chosen meal, but users can tweak it on the fly with checkboxes to include only the categories they want. Whether you eat three square meals or graze all day, this app delivers tailored, hassle-free meal ideas in seconds!

2. Who is this app for

This app is for anyone who wants control and variety in their meals—busy professionals, indecisive eaters, or creative cooks. It's perfect for users with diverse eating habits, from traditional meal planners to snack enthusiasts, who need a quick, customizable way to decide what's next on their plate.

3. Functional Requirements

- Custom Meals: Users can define and name their own meal types (e.g., "Breakfast," "Dinner," "Evening Snack").
- Custom Categories per Meal: Users can create unique categories for each meal (e.g., "Breakfast" might have "Grain" and "Fruit," while "Dinner" has "Protein" and "Side").
- Food Entry: Users can add food items to each category within a meal, with lists persisting and expandable over time.
- Storage: The app saves all meal types, categories, and food entries for future use.
- Meal Selection & Randomization: Users select a meal type; by default, pressing "Generate Meal" pulls one random item from each category. Optionally, users can check/uncheck categories to include only specific ones in the suggestion.

4. User Stories

- As a user, I want to define my own meal types and categories so the app fits my eating routine.
- As a user, I want to add and save food items to my categories so I can expand my options over time.
- As a user, I want a random meal suggestion pulling from all categories by default so I get a full idea with one tap.
- As a user, I want to choose which categories to include in a suggestion so I can skip items I don't want this time.

5. User Interface

The app will feature a simple, clean design with an intuitive, minimalist layout:

- Home Screen: A list of user-defined meal types (e.g., "Breakfast," "Dinner") with an "Add Meal" button and a dropdown or picker to select a meal.
- Meal View: For each meal, a list of custom categories with an "Add Category" button; tapping a category shows its food list with an "Add Food" option.
- Randomization Screen: After selecting a meal, a screen shows all categories with checkboxes (all checked by default) and a "Generate Meal" button. Unchecking a category excludes it from the suggestion.
- Result Screen: A clear display of the random suggestion (e.g., "Chicken + Rice" if only two categories are checked for Dinner).
- Style: Flat design with neutral colors, bold text for readability, and a smooth flow from meal selection to category customization.

User Interface Design Document: Meal Randomizer App

Layout Structure

The app adopts a modular, tile-based layout with a focus on interactivity:

- **Home Screen**: A grid of colorful meal tiles (e.g., "Breakfast," "Dinner") that users can drag to reorder. A "New Meal" tile sits prominently in the grid for quick additions.
- **Meal Customization**: A full-screen canvas where category boxes stack vertically or snap into a grid, adjustable via drag-and-drop. Food lists expand within each box when tapped.
- **Randomization Screen**: A dynamic "mixer" view taking up the full screen, with animated category spinners and a central "Spin" button for generating suggestions.

Core Components

- **Meal Tiles**: Square cards on the home screen displaying meal names, with subtle icons for quick recognition (e.g., a sun for "Breakfast").
- **Category Boxes**: Rectangular, draggable containers in the customization view, each labeled (e.g., "Protein") and expandable to show food items.
- **Spinners**: Circular, animated elements on the randomization screen, one per category, with a tap-to-lock/unlock feature.
- **Action Buttons**: "New Meal" (home), "Add Category"/"Add Food" (customization), and "Spin" (randomization) buttons, all bold and rounded for prominence.

Interaction Patterns

- **Drag-and-Drop**: Users reorder meal tiles on the home screen or arrange category boxes in the customization view.
- **Tap to Edit**: Tapping a category box reveals its food list; a "+" icon adds new items via a quick text input.
- **Spinner Control**: On the randomization screen, tapping a spinner locks/unlocks its category (locked spinners stay static during generation). Pressing "Spin" triggers an animation, then displays the result in a pop-up card.
- **Swipe**: Swipe a meal tile or category box to reveal a "Delete" option.

Visual Design Elements & Color Scheme

- **Style**: Playful flat design with rounded edges and subtle shadows for depth.
- **Colors**: A vibrant yet balanced palette—soft pastels (e.g., mint green, peach) for backgrounds, bold accents (e.g., teal, coral) for buttons and spinners, and neutral grays for text.
- **Animations**: Smooth transitions for tile drags, a lively spin effect for randomization, and a bounce-in for result pop-ups.

Mobile, Web App, Desktop Considerations

- **Mobile**: Optimized for one-handed use with larger tap targets and a condensed grid (2x2 tiles on smaller screens). Spinners stack vertically if screen height is limited.
- **Web App**: Expands the grid to 3x3 or 4x4 tiles, with category boxes in a horizontal row during customization. Drag-and-drop is mouse-friendly.
- **Desktop**: Full-screen canvas for customization with a sidebar for meal previews. Spinners align horizontally for a wider randomization view.

Typography

- **Font**: A clean, sans-serif font like "Poppins" or "Open Sans" for readability and a modern feel.
- **Sizes**: 24px for headings (meal names, category labels), 16px for body text (food items), 14px for secondary text (buttons, hints).
- **Weight**: Bold for headings and buttons, regular for lists, ensuring hierarchy and clarity.

Accessibility

- **Contrast**: High-contrast text (e.g., dark gray on pastel backgrounds) meets WCAG 2.1 AA standards.
- **Touch Targets**: Buttons and spinners are at least 48x48px for easy tapping.
- **Screen Reader Support**: Labels for all interactive elements (e.g., "Spin button," "Protein category spinner") and descriptive alt text for icons.
- **Colorblind-Friendly**: Distinct shapes and patterns (e.g., dotted outlines for locked spinners) alongside colors for differentiation.

Software Requirements Specification Document: Meal Randomizer App

System Design

- The app will be a cross-platform mobile application with optional web support.
- It will operate as a single-user system with local data storage and optional cloud sync.
- Core functionality includes meal creation, category management, food entry, and randomization, all accessible offline with sync capabilities for multi-device use.

Architecture Pattern

- **Pattern**: Model-View-Controller (MVC)
    - **Model**: Manages data (meals, categories, food items) stored locally and synced to the cloud.
    - **View**: Displays the UI (home screen, meal views, randomization screen) as per the User Interface Design Document.
    - **Controller**: Handles user inputs (e.g., adding meals, generating suggestions) and updates the model and view.
- **Reason**: MVC is straightforward for beginners, separates concerns, and supports the app's modular, interactive design.

State Management

- **Approach**: Simple state management using a reactive framework (e.g., Provider in Flutter).
- **Details**:
    - State is maintained locally for meal types, categories, and food lists.
    - UI updates reactively when state changes (e.g., adding a category or generating a meal).
    - Cloud sync updates state across devices when online.
- **Reason**: Keeps complexity low for a beginner while ensuring a responsive UI.

Data Flow

- **Input**: User creates meal types, adds categories, and enters food items via the UI.
- **Processing**: Controller processes inputs, updates the model, and triggers randomization logic.
- **Output**: View displays updated lists or randomized meal suggestions.
- **Sync**: Local data is saved instantly; cloud sync occurs when internet is available (optional).

Technical Stack

- **Language**: Dart
    - Simple syntax, beginner-friendly, and powerful for UI development.
- **Framework**: Flutter

    - Cross-platform (mobile, web, desktop) with a single codebase, rich widget library for the app's playful flat design, and good community support.
- **Database**: SQLite (local) + Firebase Firestore (cloud, optional)
    - SQLite for offline storage (meals, categories, foods); Firestore for sync across devices.
- **State Management**: Provider
    - Lightweight, easy to learn, and integrates well with Flutter.
- **Tools**: Visual Studio Code (IDE), Flutter SDK, Dart DevTools (debugging).
- **Reason**: This stack balances ease of learning with the app's functional and UI needs.

Authentication Process

- **Approach**: Optional user authentication for cloud sync.
- **Details**:
    - No authentication required for local use.
    - Optional Google Sign-In via Firebase Authentication for cloud sync.
    - Users can skip sign-in and use the app offline indefinitely.
- **Reason**: Keeps the app simple for local use while offering sync as an optional feature.

Route Design

- **Routes**:
    - /home: Home screen with meal tiles.
    - /meal/:id: Meal customization view for a specific meal (dynamic ID).
    - /randomize/:id: Randomization screen for a selected meal.
    - /result/:id: Result screen showing the generated meal suggestion.
- **Navigation**: Stack-based navigation with a back button; swipe gestures supported on mobile.
- **Reason**: Matches the app's flow from meal selection to randomization.

API Design

- **Internal APIs**: No external APIs required for core functionality.
- **Local Data API**:
    - createMeal(name): Creates a new meal type.
    - addCategory(mealId, name): Adds a category to a meal.
    - addFood(categoryId, name): Adds a food item to a category.
    - generateMeal(mealId, selectedCategories): Returns a randomized meal suggestion.
- **Cloud Sync API (Optional)**:
    - syncData(userId): Syncs local data to Firestore.
    - fetchData(userId): Retrieves synced data from Firestore.
- **Reason**: Simple CRUD operations for local use; optional sync for future scalability.

Database Design ERD

- **Entities**:
    - **Meal**:
        - mealId (Primary Key, int)
        - name (string, e.g., "Breakfast")
    - **Category**:
        - categoryId (Primary Key, int)
        - mealId (Foreign Key, int)
        - name (string, e.g., "Protein")
    - **Food**:

        - foodId (Primary Key, int)
        - categoryId (Foreign Key, int)
        - name (string, e.g., "Chicken")
- **Relationships**:
    - One Meal has many Categories (1:N).
    - One Category has many Foods (1:N).
- **Storage**: SQLite tables locally; Firestore collections if synced (e.g., meals/{mealId}/categories/{categoryId}/foods).